Machine Learning Mastery
Making Developers Awesome at Machine Learning

Step-By-Step Framework for Imbalanced Classification Projects

by **Jason Brownlee** on March 9, 2020 in **Imbalanced Classification**

Last Updated on March 19, 2020

Classification predictive modeling problems involve predicting a class label for a given set of inputs.

It is a challenging problem in general, especially if little is known about the dataset, as there are tens, if not hundreds, of machine learning algorithms to choose from. The problem is made significantly more difficult if the distribution of examples across the classes is imbalanced. This requires the use of specialized methods to either change the dataset or change the learning algorithm to handle the skewed class distribution.

A common way to deal with the overwhelm on a new classification project is to use a favorite machine learning algorithm like Random Forest or SMOTE. Another common approach is to scour the research literature for descriptions of vaguely similar problems and attempt to re-implement the algorithms and configurations that are described.

These approaches can be effective, although they are hit-or-miss and time-consuming respectively. Instead, the shortest path to a good result on a new classification task is to systematically evaluate a suite of machine learning algorithms in order to discover what works well, then double down. This approach can also be used for imbalanced classification problems, tailored for the range of data sampling, cost-sensitive, and one-class classification algorithms that one may choose from.

In this tutorial, you will discover a systematic framework for working through an imbalanced classification dataset.

After completing this tutorial, you will know:

- The challenge of choosing an algorithm for imbalanced classification.
- A high-level framework for systematically working through an imbalanced classification project.

- Specific algorithm suggestions to try at each step of an imbalanced classification project.

Discover SMOTE, one-class classification, cost-sensitive learning, threshold moving, and much more in my new book, with 30 step-by-step tutorials and full Python source code.

Let’s get started.

Step-By-Step Framework for Imbalanced Classification Projects
Photo by ~jar{}, some rights reserved.

Tutorial Overview

This tutorial is divided into three parts; they are:

1. What Algorithm To Use?
2. Use a Systematic Framework
3. Detailed Framework for Imbalanced Classification
    1. Select a Metric
    2. Spot Check Algorithms
    3. Spot Check Imbalanced Algorithms
    4. Hyperparameter Tuning

1. What Algorithm To Use?

You’re handed or acquire an imbalanced classification dataset. *Now what?*

There are so many machine learning algorithms to choose from, let alone techniques specifically designed for imbalanced classification.

Which algorithms do you use? *How do you choose?*

This is the challenge faced at the beginning of each new imbalanced classification project. It is this challenge that makes applied machine learning both thrilling and terrifying.

There are perhaps two common ways to solve this problem:

- Use a favorite algorithm.
- Use what has worked previously.

One approach might be to select a favorite algorithm a t
approach to a solution but is only effective if your favo for
your specific dataset.

Another approach might be to review the literature and ts
like yours. This can be effective if many people have s

In practice, this is rarely the case, and research public pet
algorithm rather than presenting an honest compariso
ideas for techniques to try.

Instead, if little is known about the problem, then the shortest path to a “*good*” result is to systematically test a suite of different algorithms on your dataset.

2. Use a Systematic Framework

Consider a balanced classification task.

You’re faced with the same challenge of selecting which algorithms to use to address your dataset.

There are many solutions to this problem, but perhaps the most robust is to systematically test a suite of algorithms and use empirical results to choose.

Biases like "*my favorite algorithm*" or "*what has worked in the past*" can feed ideas into the study, but can lead you astray if relied upon. Instead, you need to let the results from systematic empirical experiments to tell you what algorithm is good or best for your imbalanced classification dataset.

Once you have a dataset, the process involves the three steps of (1) selecting a metric by which to evaluate candidate models, (2) testing a suite of algorithms, and (3) tuning the best performing models. This may not the only approach; it is just the simplest reliable process to get you from "*I have a new dataset*" to "*I have good results*" very quickly.

This process can be summarized as follows:

1. Select a Metric
2. Spot Check Algorithms
3. Hyperparameter Tuning

Spot-checking algorithms is a little more involved as m ion such as scaling, removal of outliers, and more. Also, e ul design of a test harness, often involving the use of k-f f a given model on unseen data.

We can use this simple process for imbalanced classification.

It is still important to spot check standard machine learning algorithms on imbalanced classification. Standard algorithms often do not perform well when the class distribution is imbalanced. Nevertheless, testing them first provides a baseline in performance by which more specialized models can be compared and must out-perform.

It is also still important to tune the hyperparameters of well-performing algorithms. This includes the hyperparameters of models specifically designed for imbalanced classification.

Therefore, we can use the same three-step procedure and insert an additional step to evaluate imbalanced classification algorithms.

We can summarize this process as follows:

1. Select a Metric
2. Spot Check Algorithms
3. Spot Check Imbalanced Algorithms
4. Hyperparameter Tuning

This provides a high-level systematic framework to work through an imbalanced classification problem.

Nevertheless, there are many imbalanced algorithms to choose from, let alone many different standard machine learning algorithms to choose from.

We require a similar low-level systematic framework for each step.

3. Detailed Framework for Imbalanced Classification

We can develop a similar low-level framework to systematically work through each step of an imbalanced classification project.

From selecting a metric to hyperparameter tuning.

3.1. Select a Metric

Selecting a metric might be the most important step in the project.

The metric is the measuring stick by which all models a wrong metric can mean choosing the wrong algorithm rom the problem you actually want solved.

The metric must capture those details about a model o oject or project stakeholders.

This is hard, as there are many metrics to choose fron t they want. There may also be multiple ways to frame t few different framings and, in turn, different metrics to see

First, you must decide whether you want to predict probabilities or crisp class labels. Recall that for binary imbalanced classification tasks, the majority class is normal, called the "*negative class*", and the minority class is the exception, called the "*positive class*".

Probabilities capture the uncertainty of the prediction, whereas crisp class labels can be used immediately.

- **Probabilities**: Predict the probability of class membership for each example.
- **Class Labels**: Predict a crisp class label for each example.

3.1.1. Predict Probabilities

If probabilities are intended to be used directly, then a good metric might be the Brier Score and the Brier Skill score.

Alternately, you may want to predict probabilities and allow the user to map them to crisp class labels themselves via a user-selected threshold. In this case, a measure can be chosen that summarizes the performance of the model across the range of possible thresholds.

If the positive class is the most important, then the precision-recall curve and area under curve (PR AUC) can be used. This will optimize both precision and recall across all thresholds.

Alternately, if both classes are equally important, the ROC Curve and area under curve (ROC AUC) can be used. This will maximize the true positive rate and minimize the false positive rate.

3.1.2. Predict Class Labels

If class labels are required and both classes are equally important, a good default metric is classification accuracy. This only makes sense if the majority class is less than about 80 percent off the data. A majority class that has a greater than 80 percent or 90 percent skew will swamp the accuracy metric and it will lose its meaning for comparing algorithms.

If the class distribution is severely skewed, then the G-mean metric can be used that will optimize the sensitivity and specificity metrics.

If the positive class is more important, then variations of the F-Measure can be used that optimize the precision and recall. If both false positive and false negatives are equally important, then F1 can be used. If false negatives are more costly, then the F2-Measure can be used, otherwise, if false positives are more costly, then the F0.5-Measure can be used.

3.1.3. Framework for Choosing a Metric

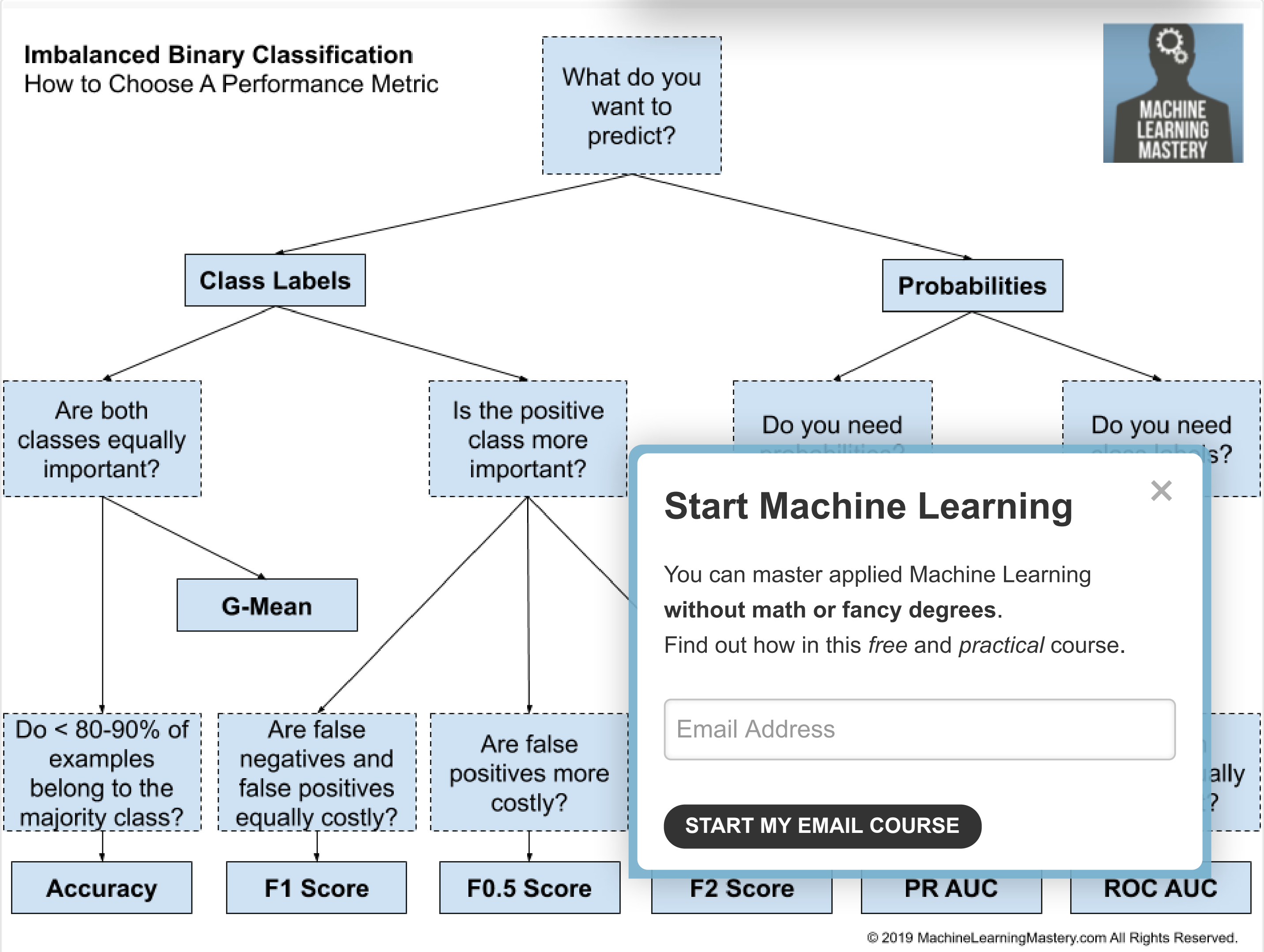
These are just heuristics but provide a useful starting p

imbalanced classification task.

We can summarize these heuristics into a framework a

- **Are you predicting probabilities?**
  - **Do you need class labels?**
    - **Is the positive class more important?**
      - Use Precision-Recall AUC
    - **Are both classes important?**
      - Use ROC AUC
  - **Do you need probabilities?**
    - Use Brier Score and Brier Skill Score
- **Are you predicting class labels?**
  - **Is the positive class more important?**
    - **Are False Negatives and False Positives Equally Costly?**
      - Use F1-Measure
    - **Are False Negatives More Costly?**
      - Use F2-Measure
    - **Are False Positives More Costly?**
      - Use F0.5-Measure
  - **Are both classes important?**
    - **Do you have < 80%-90% Examples for the Majority Class?**
      - Use Accuracy
    - **Do you have > 80%-90% Examples for the Majority Class?**
      - Use G-Mean

We can also transform these decisions into a decision tree, as follows.


Imbalanced Binary Classification
How to Choose A Performance Metric
What do you want to predict?
MACHINE LEARNING MASTERY
Class Labels
Probabilities
Are both classes equally important?
Is the positive class more important?
Do you need probabilities?
Do you need class labels?
G-Mean
Do < 80-90% of examples belong to the majority class?
Are false negatives and false positives equally costly?
Are false positives more costly?
ally ?
Accuracy
F1 Score
F0.5 Score
F2 Score
PR AUC
ROC AUC
© 2019 MachineLearningMastery.com All Rights Reserved.

How to Choose a Metric for Imbalanced Classification

Once a metric has been chosen, you can start evaluating machine learning algorithms.

3.2. Spot Check Algorithms

Spot checking machine learning algorithms means evaluating a suite of different types of algorithms with minimal hyperparameter tuning.

Specifically, it means giving each algorithm a good chance to learn about the problem, including performing any required data preparation expected by the algorithm and using best-practice configuration options or defaults.

The objective is to quickly test a range of standard machine learning algorithms and provide a baseline in performance to which techniques specialized for imbalanced classification must be compared and outperform in order to be considered skillful. The idea here is that there is little point in using fancy imbalanced algorithms if they cannot out-perform so-called unbalanced algorithms.

A robust test harness must be defined. This often involves k-fold cross-validation, often with k-10 as a sensible default. Stratified cross-validation is often required to ensure that each fold has the same class distribution as the original dataset. And the cross-validation procedure is often repeated multiple times, such as 3, 10, or 30 in order to effectively capture a sample of model performance on the dataset, summarized with a mean and standard deviation of the scores.

There are perhaps four levels of algorithms to spot check; they are:

1. Naive Algorithms
2. Linear Algorithms
3. Nonlinear Algorithms
4. Ensemble Algorithms

3.2.1. Naive Algorithms

Firstly, a naive classification must be evaluated.

This provides a rock-bottom baseline in performance t e
skill on the dataset.

Naive means that the algorithm has no logic other thar he
choice of naive algorithm is based on the choice of pe

For example, a suitable naive algorithm for classificati
cases. A suitable naive algorithm for the Brier Score w
probability of each class in the training dataset.

A suggested mapping of performance metrics to naive algorithms is as follows:

- **Accuracy**: Predict the majority class (class 0).
- **G-Mean**: Predict a uniformly random class.
- **F-Measure**: Predict the minority class (class 1).
- **ROC AUC**: Predict a stratified random class.
- **PR ROC**: Predict a stratified random class.
- **Brier Score**: Predict majority class prior.

If you are unsure of the "*best*" naive algorithm for your metric, perhaps test a few and discover which results in the better performance that you can use as your rock-bottom baseline.

Some options include:

- Predict the majority class in all cases.
- Predict the minority class in all cases.
- Predict a uniform randomly selected class.
- Predict a randomly selected class selected with the prior probabilities of each class.
- Predict the class prior probabilities.

3.2.2. Linear Algorithms

Linear algorithms are those that are often drawn from the field of statistics and make strong assumptions about the functional form of the problem.

We can refer to them as linear because the output is a linear combination of the inputs, or weighted inputs, although this definition is stretched. You might also refer to these algorithms as probabilistic algorithms as they are often fit under a probabilistic framework.

They are often fast to train and often perform very well. Examples of linear algorithms you should consider trying include:

- Logistic Regression
- Linear Discriminant Analysis
- Naive Bayes

3.2.3. Nonlinear Algorithms

Nonlinear algorithms are drawn from the field of machi

functional form of the problem.

We can refer to them as nonlinear because the output

They often require more data than linear algorithms ar

algorithms you should consider trying include:

- Decision Tree
- k-Nearest Neighbors
- Artificial Neural Networks
- Support Vector Machine

3.2.4. Ensemble Algorithms

Ensemble algorithms are also drawn from the field of machine learning and combine the predictions from two or more models.

There are many ensemble algorithms to choose from, but when spot-checking algorithms, it is a good idea to focus on ensembles of decision tree algorithms, given that they are known to perform so well in practice on a wide range of problems.

Examples of ensembles of decision tree algorithms you should consider trying include:

- Bagged Decision Trees
- Random Forest
- Extra Trees
- Stochastic Gradient Boosting

3.2.5. Framework for Spot-Checking Machine Learning Algorithms

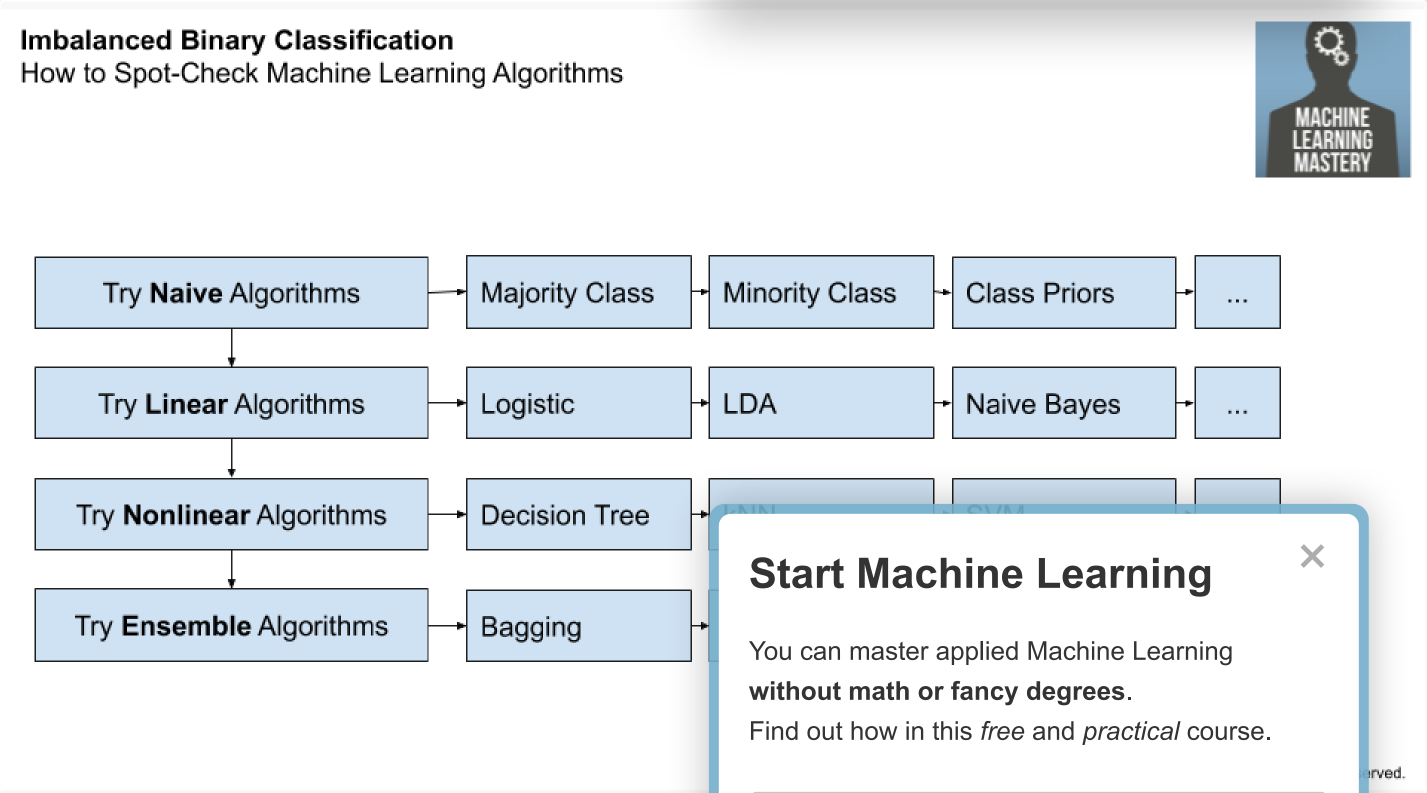
We can summarize these suggestions into a framework for testing machine learning algorithms on a dataset.

- Naive Algorithms
  - Majority Class
  - Minority Class
  - Class Priors
- Linear Algorithms
  - Logistic Regression
  - Linear Discriminant Analysis
  - Naive Bayes
- Nonlinear Algorithms
  - Decision Tree
  - k-Nearest Neighbors
  - Artificial Neural Networks
  - Support Vector Machine
- Ensemble Algorithms
  - Bagged Decision Trees
  - Random Forest
  - Extra Trees
  - Stochastic Gradient Boosting

The order of the steps is probably not flexible. Think o ity, and in turn, capability.

The order of algorithms within each step is flexible and the list of algorithms is not complete, and probably never could be given the vast number of algorithms available. Limiting the algorithms tested to a subset of the most common or most widely used is a good start. Using data-preparation recommendations and hyperparameter defaults is also a good start.

The figure below summarizes this step of the framework.



How to Spot-Check Mac

3.3. Spot Check Imbalanced Algorithm

Spot-checking imbalanced algorithms is much like spot-checking machine learning algorithms.

The objective is to quickly test a large number of techniques in order to discover what shows promise so that you can focus more attention on it later during hyperparameter tuning.

The spot-checking performed in the previous section provides both naive and modestly skillful models by which all imbalanced techniques can be compared. This allows you to focus on these methods that truly show promise on the problem, rather than getting excited about results that only appear effective compared only to other imbalanced classification techniques (which is an easy trap to fall into).

There are perhaps four types of imbalanced classification techniques to spot check:

1. Data Sampling Algorithms
2. Cost-Sensitive Algorithms
3. One-Class Algorithms
4. Probability Tuning Algorithms

3.3.1. Data Sampling Algorithms

Data sampling algorithms change the composition of the training dataset to improve the performance of a standard machine learning algorithm on an imbalanced classification problem.

There are perhaps three main types of data sampling techniques; they are:

- Data Oversamplinug.
- Data Undersampling.
- Combined Oversampling and Undersampling.

Data oversampling involves duplicating examples of the minority class or synthesizing new examples from the minority class from existing examples. Perhaps the most popular methods is SMOTE and variations such as Borderline SMOTE. Perhaps the most important hyperparameter to tune is the amount of oversampling to perform.

Examples of data oversampling methods include:

- Random Oversampling
- SMOTE
- Borderline SMOTE
- SVM SMote
- k-Means SMOTE
- ADASYN

Undersampling involves deleting examples from the m thm to carefully choose which examples to delete. Popular neighbors and Tomek links.

Examples of data undersampling methods include:

- Random Undersampling
- Condensed Nearest Neighbor
- Tomek Links
- Edited Nearest Neighbors
- Neighborhood Cleaning Rule
- One-Sided Selection

Almost any oversampling method can be combined with almost any undersampling technique. Therefore, it may be beneficial to test a suite of different combinations of oversampling and undersampling techniques.

Examples of popular combinations of over and undersampling include:

- SMOTE and Random Undersampling
- SMOTE and Tomek Links
- SMOTE and Edited Nearest Neighbors

Data sampling algorithms may perform differently depending on the choice of machine learning algorithm.

As such, it may be beneficial to test a suite of standard machine learning algorithms, such as all or a subset of those algorithms used when spot checking in the previous section.

Additionally, most data sampling algorithms make use of the k-nearest neighbor algorithm internally. This algorithm is very sensitive to the data types and scale of input variables. As such, it may be important to at least normalize input variables that have differing scales prior to testing the methods, and perhaps using specialized methods if some input variables are categorical instead of numerical.

3.3.2. Cost-Sensitive Algorithms

Cost-sensitive algorithms are modified versions of machine learning algorithms designed to take the differing costs of misclassification into account when fitting the model on the training dataset.

These algorithms can be effective when used on imbalanced classification, where the cost of misclassification is configured to be inversely proportional to the distribution of examples in the training dataset.

There are many cost-sensitive algorithms to choose from, although it might be practical to test a range of cost-sensitive versions of linear, nonlinear, and ensem

Some examples of machine learning algorithms that c
include:

- Logistic Regression
- Decision Trees
- Support Vector Machines
- Artificial Neural Networks
- Bagged Decision Trees
- Random Forest
- Stochastic Gradient Boosting

3.3.3. One-Class Algorithms

Algorithms used for outlier detection and anomaly detection can be used for classification tasks.

Although unusual, when used in this way, they are often referred to as one-class classification algorithms.

In some cases, one-class classification algorithms can be very effective, such as when there is a severe class imbalance with very few examples of the positive class.

Examples of one-class classification algorithms to try include:

- One-Class Support Vector Machines
- Isolation Forests
- Minimum Covariance Determinant
- Local Outlier Factor

3.3.4. Probability Tuning Algorithms

Predicted probabilities can be improved in two ways; they are:

- Calibrating Probabilities.
- Tuning the Classification Threshold.

Calibrating Probabilities

Some algorithms are fit using a probabilistic framework and, in turn, have calibrated probabilities.

This means that when 100 examples are predicted to have the positive class label with a probability of 80 percent, then the algorithm will predict the correct class label 80 percent of the time.

Calibrated probabilities are required from a model to be considered skillful on a binary classification task when probabilities are either required as the output or used to evaluate the model (e.g. ROC AUC or PR AUC).

Some examples of machine learning algorithms that predict calibrated probabilities are as follows:

- Logistic Regression
- Linear Discriminant Analysis
- Naive Bayes
- Artificial Neural Networks

Most nonlinear algorithms do not predict calibrated pro st-
process the predicted probabilities in order to calibrate

Therefore, when probabilities are required directly or a
algorithms are being used, it is important to calibrate t

Some examples of probability calibration algorithms to try include:

- Platt Scaling
- Isotonic Regression

Tuning the Classification Threshold

Some algorithms are designed to naively predict probabilities that later must be mapped to crisp class labels.

This is the case if class labels are required as output for the problem, or the model is evaluated using class labels.

Examples of probabilistic machine learning algorithms that predict a probability by default include:

- Logistic Regression
- Linear Discriminant Analysis
- Naive Bayes
- Artificial Neural Networks

Probabilities are mapped to class labels using a threshold probability value. All probabilities below the threshold are mapped to class 0, and all probabilities equal-to or above the threshold are mapped to class 1.

The default threshold is 0.5, although different thresholds can be used that will dramatically impact the class labels and, in turn, the performance of a machine learning model that natively predicts probabilities.

As such, if probabilistic algorithms are used that natively predict a probability and class labels are required as output or used to evaluate models, it is a good idea to try tuning the classification threshold.

3.3.5. Framework for Spot-Checking Imbalanced Algorithms

We can summarize these suggestions into a framework for testing imbalanced machine learning algorithms on a dataset.

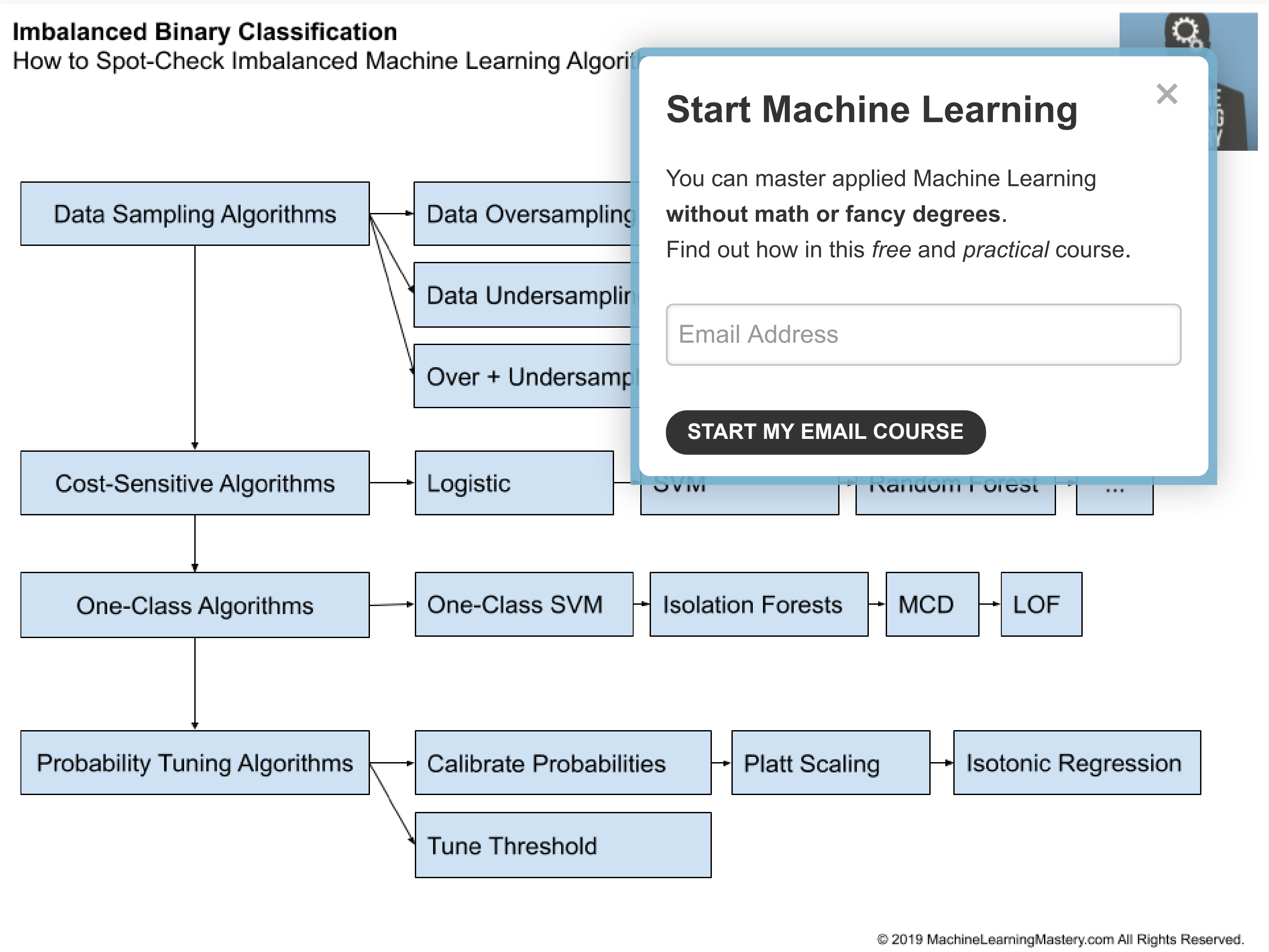
1. Data Sampling Algorithms
    - Data Oversampling
        - Random Oversampling
        - SMOTE
        - Borderline SMOTE
        - SVM SMote
        - k-Means SMOTE
        - ADASYN
    - Data Undersampling
        - Random Undersampling
        - Condensed Nearest Neighbor
        - Tomek Links
        - Edited Nearest Neighbors
        - Neighborhood Cleaning Rule
        - One Sided Selection
    - Combined Oversampling and Undersampling
        - SMOTE and Random Undersampling
        - SMOTE and Tomek Links
        - SMOTE and Edited Nearest Neighbors
2. Cost-Sensitive Algorithms
    - Logistic Regression
    - Decision Trees
    - Support Vector Machines
    - Artificial Neural Networks
    - Bagged Decision Trees
    - Random Forest
    - Stochastic Gradient Boosting
3. One-Class Algorithms
    - One-Class Support Vector Machines
    - Isolation Forests
    - Minimum Covariance Determinant
    - Local Outlier Factor
4. Probability Tuning Algorithms
    - Calibrating Probabilities

    - Platt Scaling
    - Isotonic Regression
- Tuning the Classification Threshold

The order of the steps is flexible, and the order of algorithms within each step is also flexible, and the list of algorithms is not complete.

The structure is designed to get you thinking systematically about what algorithm to evaluate.

The figure below summarizes the framework.



How to Spot-Check Imbalanced Machine Learning Algorithms

3.4. Hyperparameter Tuning

After spot-checking machine learning algorithms and imbalanced algorithms, you will have some idea of what works and what does not on your specific dataset.

The simplest approach to hyperparameter tuning is to select the top five or 10 algorithms or algorithm combinations that performed well and tune the hyperparameters for each.

There are three popular hyperparameter tuning algorithms that you may choose from:

- Random Search
- Grid Search
- Bayesian Optimization

A good default is grid search if you know what hyperparameter values to try, otherwise, random search should be used. Bayesian optimization should be used if possible but can be more challenging to set up and run.

Tuning the best performing methods is a good start, but not the only approach.

There may be some standard machine learning algorithms that perform well but do not perform as well when used with data sampling or probability calibratio n their imbalanced-classification augmentations to see i

Additionally, there may be imbalanced-classification al results in a dramatic lift in performance for one or more provide an interesting basis for further tuning to see if a

Further Reading

This section provides more resources on the topic if yo

Tutorials

- Applied Machine Learning Process
- How to Use a Machine Learning Checklist to Get Accurate Predictions, Reliably

Books

- Learning from Imbalanced Data Sets, 2018.
- Imbalanced Learning: Foundations, Algorithms, and Applications, 2013.

Summary

In this tutorial, you discovered a systematic framework for working through an imbalanced classification dataset.

Specifically, you learned:

- The challenge of choosing an algorithm for imbalanced classification.
- A high-level framework for systematically working through an imbalanced classification project.
- Specific algorithm suggestions to try at each step of an imbalanced classification project.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

Tweet Share in Share

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.
View all posts by Jason Brownlee →

‹ Imbalanced Classification with the Adult Income Dataset

Imbalanced Classification with the Fraudulent Credit Card Transactions Dataset ›

22 Responses to *Step-By-Step Framework for Imbalanced Classification Projects*

Eduardo Rojas March 9, 2020 at 9:04 am # REPLY ↩

OMG. It was really amazing. Thanks Jason

Jason Brownlee March 9, 2020 at 11:06 am # REPLY

Thanks, I’m happy it helps.

Asif Ameer March 10, 2020 at 5:30 am # REPLY

Jason you are doing tremendous Job. Really Appreciate your Efforts

Jason Brownlee March 10, 2020 at 5:43 am # REPLY

Thanks.

Pawel Rolbiecki March 9, 2020 at 10:41 pm #

Hi Jason.
Nice work! I finally found a good post about how to cho
question is about regularization and detect/avoiding ov

What are good practices to apply regularization for imb
classification. I just guess it would rather be a data set augmentation to generate more similar images. To make the dataset more “balanced” rather than focusing on tuning regularization parameters like L2 or dropout reatio. Is it a good approach?
Pawel

Jason Brownlee March 10, 2020 at 5:40 am # REPLY

I don’t follow your question, sorry. You use regularization on a model, not on a dataset.

Layne March 11, 2020 at 1:40 am # REPLY

“This only makes sense if the majority class is less than about 80 percent off the data”
haha that is all I was looking for.

Jason Brownlee March 11, 2020 at 5:24 am # REPLY

Great!

archit garg March 13, 2020 at 5:25 am # REPLY ↰

Hi Jason,

I am dealing with a multi-class(5 classes) imbalanced dataset with metrics as F1 score averaged, and applying xgboost model with random search, but I am not getting a good score, best I've achieved is 52.5. I even tried Deep Neural Network with dropouts and Smote but still no success. I tried other models Text classification like Random Forest, Gradient Boosting, Logistic Regression One vs rest, Catboost but still I am not able to achieve higher
score. Can you please suggest what else I can try.

Jason Brownlee March 13, 2020 at 8:22

The above framework is my best advi

After that, more general suggestions here:
http://machinelearningmastery.com/machine-le

amarillo March 11, 2020 at 12:50 am #

great work! very very helpful. thank you .

Jason Brownlee March 11, 2020 at 5:24 am # REPLY ↰

Thanks, I'm happy it helps.

Ganesh Prasad March 13, 2020 at 4:15 pm # REPLY ↰

Hi Jason. I have a multi-class classification problem where the number of classes is around 75. I am using neural network for the classification, but the maximum probability I'm getting after the softmax activation function at the last layer is less than 0.5. Is it fine to predict a class having the maximum probability, even if it's probability is lower than 0.5?

Jason Brownlee March 14, 2020 at 8:05 am # REPLY ↰

I would recommend choosing a performance metric to evaluate the model and use that to judge whether the probabilities are appropriate for prediction.

This might help:
https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/

Malina March 13, 2020 at 7:53 pm # REPLY

Dear Jason,

thank you very much for your awesome work. If I could give a bit of advice then it would be not to repeat yourself in the article. The shorter the article the better.

Jason Brownlee March 14, 2020 at 8:09 am #

Thanks for the advice.

Abdelrahim March 13, 2020 at 10:59 pm #

Thanks jason
It would you kind if give us tutorial about Brier Score

Jason Brownlee March 14, 2020 at 8:12 am # REPLY

Here is a tutorial on brier score:
https://machinelearningmastery.com/probability-metrics-for-imbalanced-classification/

zhaowei March 18, 2020 at 2:25 pm # REPLY

“Three may also be multiple ways to frame the problem”

Caught a typo~

Jason Brownlee March 19, 2020 at 6:21 am # REPLY

Thanks, fixed.

Patrycja March 28, 2020 at 3:19 am #

Hi! Great article! It’s interesting and insightful. We’ve learned a lot. Thank you for your work and for spreading knowledge about Machine Learning. Just wanted to let you know, that we included this piece in our Weekly Roundup on the Neptune.ai blog. Cheers!

Jason Brownlee March 28, 2020 at 6:26 am # REPLY

Thanks.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

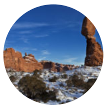
SUBMIT COMMENT

Welcome!
My name is *Jason Brownlee* PhD, and I **help developers** get results with **machine learning**.
Read more

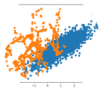
Never miss a tutorial:

Picked for you:

8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset

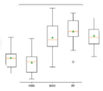


Imbalanced Classification With Python (7-Day Mini-Course)



SMOTE for Imbalanced Classification with Python

Tour of Evaluation Metrics for Imbalanced Classifica



Imbalanced Multiclass Classification with the Glass I

Loving the

The Imbalanced Classification with Python E

SEE WHAT'S INSIDE

Address: PO Box 206, Vermont Victoria 3133, Australia. | ACN: 626 223 336.
LinkedIn | Twitter | Facebook | Newsletter | RSS

Privacy | Disclaimer | Terms | Contact | Sitemap | Search